

Computer Architecture and Performance

◆ **Last Time**

- Background: Boolean Algebra, Logic, and C/C++ programming
- Architecture = Interfaces, System level structure

◆ **Today**

- Quiz #1
- Computer Architecture
- Basic Performance

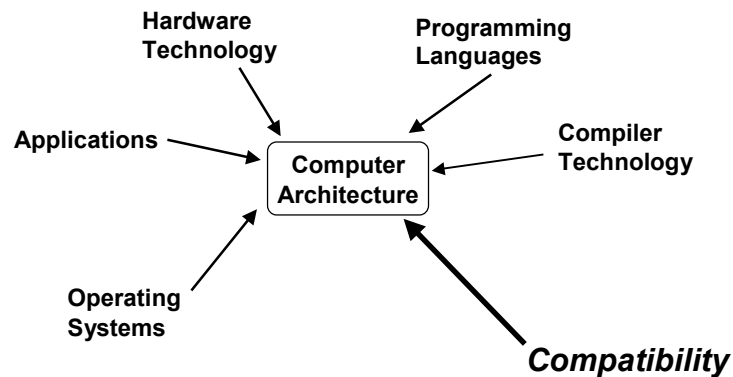
◆ **Reminders/Announcements**

- TA Office Hours and Location set (see the web)
 - » Wed: 9:00-11:00AM, AP&M 3337D
 - » Fri: 10:00-11:00AM, AP&M 3337D
- Reading: Chapter II, Patterson & Hennessy
- Homework: HW #1 will be posted on the web tomorrow

CS 141 Chien

Jan 13, 2000

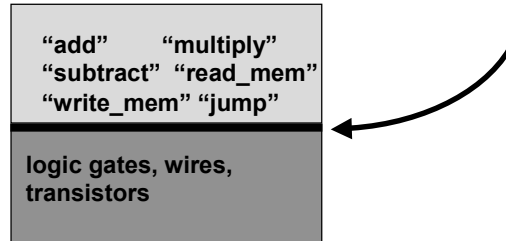
Forces on Computer Architecture



CS 141 Chien

Jan 13, 2000

The Instruction Set Architecture



- ◆ Classical definition of computer architecture,
- ◆ Part that is visible to the programmer; persists over architecture *implementations*
 - opcodes (available instructions)
 - number and types of registers
 - instruction formats
 - storage access, addressing modes
 - exceptional conditions

CS 141 Chien

Jan 13, 2000

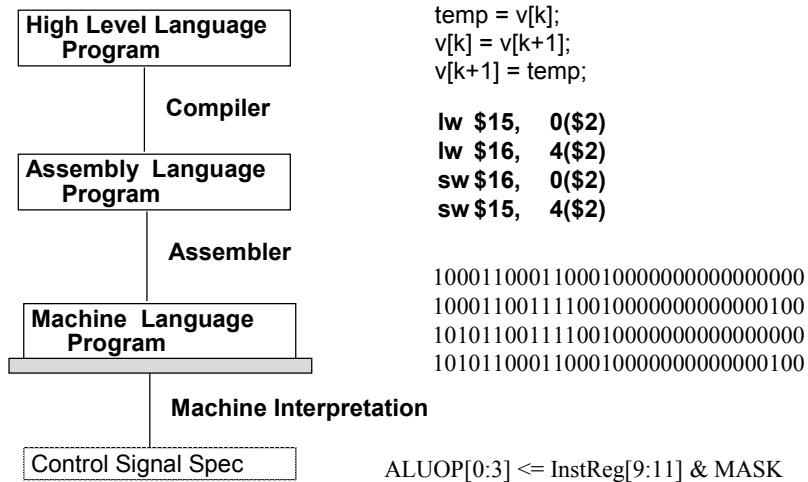
Examples of ISAs

- ◆ Alpha AXP
- ◆ 80x86 (486, Pentium, Pentium Pro, Pentium II+MMX, Pentium III+KNI, AMD K6, AMD Athlon)
- ◆ PowerPC 601, 603, G3, G4
- ◆ VAX
- ◆ MIPS
- ◆ SPARC
- ◆ IBM 360
- ◆ ...

CS 141 Chien

Jan 13, 2000

How to Speak Computer



CS 141 Chien

Jan 13, 2000

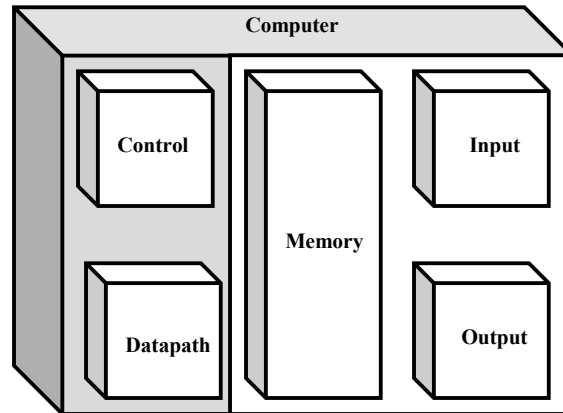
The Challenge of Computer Architecture

- ◆ Once you have decided on an ISA, you must decide
 - how to design the hardware to execute it as fast as possible
- ◆ Redone for every new implementation, technology constraints vary
- ◆ The industry changes faster than any other.
- ◆ The ground rules change every year.
 - new problems
 - new opportunities
 - different tradeoffs
- ◆ Challenge: making programs run faster
 - Run with less power
 - Run in a smaller/lighter system
 - ...

CS 141 Chien

Jan 13, 2000

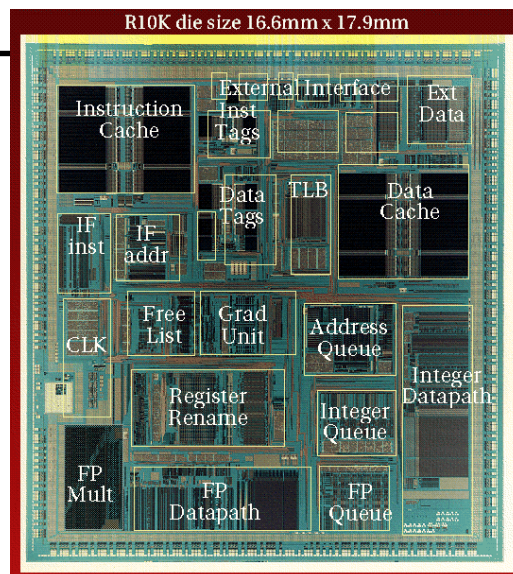
Five Classic Components



CS 141 Chien

Jan 13, 2000

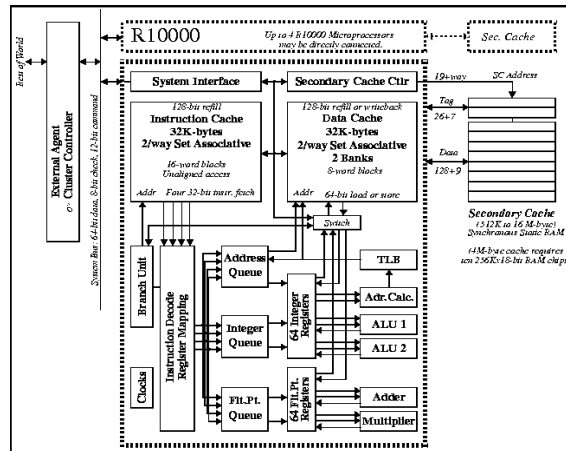
MIPS R10000 CPU



CS 141 Chien

Jan 13, 2000

MIPS R10000 CPU



CS 141 Chien

Jan 13, 2000

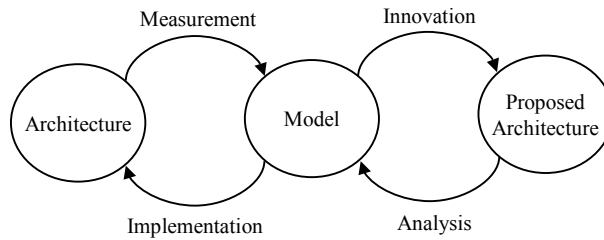
Computer Architecture

- ◆ Design of persistent interfaces which define the structure of software and hardware
 - Instruction set architecture
 - Processor Bus
- ◆ Design of implementations which efficiently implement the interfaces
 - Pentium II chip
 - 440FX system chip set
 - ...

CS 141 Chien

Jan 13, 2000

Performance Measurement and Analysis in Computer Architecture

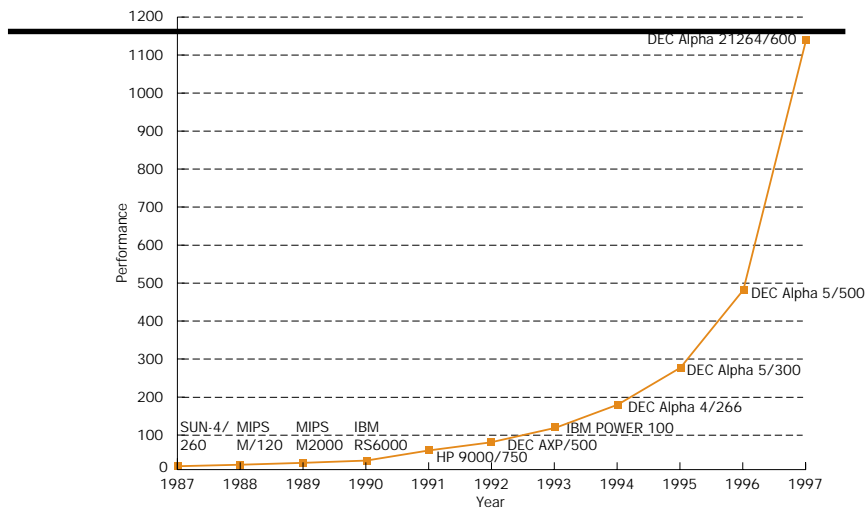


◆ Design drives implementation which informs design (and the rules keep changing)

CS 141 Chien

Jan 13, 2000

Performance Marches On ...



◆ But what is performance?

CS 141 Chien

Jan 13, 2000

The bottom line: Performance

Plane	DC to Paris time	Speed	Passengers	Throughput (p x mph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

- Time to do the task
 - *execution time*, response time, latency
- Tasks per day, hour, week, sec, ns. ...
 - *throughput*, bandwidth

CS 141 Chien

Jan 13, 2000

Measuring Performance

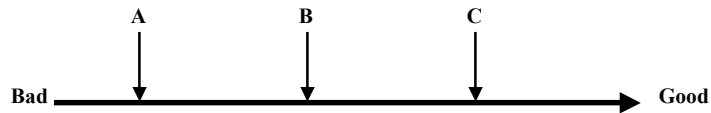


- ◆ How is performance measured?
- ◆ How to choose amongst different machines?
 - Cost: price, technology metrics
 - Performance
 - » Computing is interchangeable
 - » Computed on RED = computed on GREEN
 - » metrics capture time and speed, indicate relative performance
 - » Runtime or X's per second

CS 141 Chien

Jan 13, 2000

Performance Metrics



- ◆ What is a metric?
 - Basis for measuring
 - Basis for comparison
- ◆ Examples
 - Job: Salary, Responsibilities; School: Grades;
 - Mutual Funds: Total Return, Risk
 - Cars: Top speed, MPG's, Acceleration, Impact test
- ◆ Metrics are used to Order and Compare
- ◆ Many metrics are possible, many used

CS 141 Chien

Jan 13, 2000

Computer Performance Metrics

- ◆ Execution Time (CPU time, wall clock time)
- ◆ Millions of Instructions Per Second (MIPS)
 - Machine instructions (small primitive units generated by the compiler)
- ◆ Cycles Per Instruction (CPI)
 - Fixed length time periods, normalization for technology
- ◆ Clock Rate (Megahertz)
 - Millions of cycles per second
- ◆ => Many metrics are used.
 - Some better, some worse
- ◆ Goal: use metrics which reflect performance delivered to real user programs, real applications.

CS 141 Chien

Jan 13, 2000

Execution Time

◆ Steps

- Select a program
- Execute the program
- Measure the CPU time, or the wall clock (stopwatch) time

$$\text{Performance} = 1 / (\text{Execution time})$$

$$\text{work/sec} = 1 / \text{secs (units)}$$

◆ Smaller execution time -> better performance

◆ Larger execution time -> worse performance

◆ Limitations?

- Different programs
- Measuring time

CS 141 Chien

Jan 13, 2000

Pitfalls of Execution Time

◆ Different Programs

- Would like to run EXACTLY the same program, but...
- Programs are large and unwieldy, data is critical, and they and their usage change over time
- Need to run entire application programs at full data set sizes to get good performance information
- Best predictors, tricky, difficult, sometimes misleading

◆ “Benchmarking”

- How applications will really run
- Lies, Damn lies, and Benchmarks
- Basis of comparison amongst commercial systems: SPECmarks, TPC, Dhrystones, etc.

CS 141 Chien

Jan 13, 2000

Measuring Time

◆ **How to measure execution time?**

- Stopwatch (wall clock)
- **Computer timing: User (processor time) + System (operating system -- I/O, etc.)**
- Other users? Measurement errors?
- Compare based on processor time for processor performance
- Important, but of decreasing importance for SYSTEM performance.

◆ **Boundaries are often blurry....**

◆ **Most of class, we'll focus on processor time.**

CS 141 Chien

Jan 13, 2000

What is Time?

$$\begin{aligned}\text{CPU Execution Time} &= \text{CPU clock cycles} * \text{Clock cycle time} \\ &= \text{CPU clock cycles} / \text{Clock rate}\end{aligned}$$

- ◆ **Every conventional processor has a clock with an associated clock cycle time or clock rate.**
- ◆ **Every program runs in an integral number of clock cycles.**

x MHz = x millions of cycles/second (clock rate)

1/ (x MHz) = cycle time, 1/(500 MHz) = 2 ns

CS 141 Chien

Jan 13, 2000

MIPS and MFLOPS

◆ **MIPS - million instructions per second**

$$= \frac{\text{number of instructions executed in program}}{\text{execution time in seconds} * 10^6} = \frac{\text{Clock rate}}{\text{CPI} * 10^6}$$

◆ **MFLOPS - million floating point operations per second**

$$= \frac{\text{number of floating point operations executed in program}}{\text{execution time in seconds} * 10^6}$$

CS 141 Chien

Jan 13, 2000

Millions of Instructions per Second (MIPS)

◆ $\text{MIPS} = \frac{\text{\# of insts}}{\text{Time} * 10^6} \quad (\text{insts/sec})$

◆ **All rates measures of performance are:**

- $\frac{\text{Units of work}}{\text{Time Unit}} = \frac{X's}{\text{Sec}}$

◆ **Problem: to make these measures representative, units of work must be conserved.**

- They must correspond to real work that is irreducible!
- (i.e. work that is conserved over ANY implementation)

CS 141 Chien

Jan 13, 2000

Units of Work

- ◆ Instructions, Floating Point Operations, Window updates, answers, etc.
- ◆ Are these things conserved in a computation?
- ◆ Instructions: compiler, architecture
- ◆ Floating Point operations: compiler, algorithm
- ◆ Window updates: algorithm
- ◆ Answers to real problems: ??
- ◆ Depends on compiler, architecture, algorithm, implementation, etc. => all of this is part of the benchmark

CS 141 Chien

Jan 13, 2000

Example: Instructions are not always conserved

```
load  R1, 16(R2)      addi  R1, R1, 1
add   R3, R4, R1      load  R2, 16(R1)
load  R1, 16(R2)      addi  R1, R1, 1
add   R5, R6, R1      load  R3, 16(R1)
                                addi  R1, R1, 1
                                load  R4, 16(R1)
```

- ◆ Loads and stores may be redundant
 - data motion, not computation, real work
- ◆ Arithmetic operations may be redundant
 - 3 adds can be reduced to one
 - add 3, and fix all of the other offsets
- ◆ => Many things which seem like work can be optimized away...

CS 141 Chien

Jan 13, 2000

Example: Floating point operations are not always conserved

- ◆ **Matrix multiplication: Strassen's algorithm**
 - Algorithmic improvements
- ◆ **Iterative algorithms that converge, different precision or subtle arithmetic differences can have a major effect.**
 - Precision, arithmetic details
- ◆ **Errors (flaws) can require workarounds**
 - Intel Pentium bug, numerous operations to replace each FDIV (multiple floating point operations)
- ◆ **Others?**

CS 141 Chien

Jan 13, 2000

A Benchmarking Example

- ◆ **Pentium III 750Mhz system, Microsoft C++ compiler**
 - Compile the program, execute, count instructions
 - Measure at 800 MIPs
- ◆ **What does this tell you about performance?**
- ◆ **Compile again, this time with optimization ON!**
 - Compile takes a lot longer, execute, count instructions
 - Measure performance at 650 MIPs
- ◆ **What happened?**

CS 141 Chien

Jan 13, 2000

Benchmarking Example (cont.)

$$\frac{\# \text{ of Insts}_A}{\text{ExecTime}_A} \quad \text{vs} \quad \frac{\# \text{ of Insts}_B}{\text{ExecTime}_B}$$

- ◆ There are fewer instructions executed in the optimized program!
- ◆ MIPS rating depends on compiler
 - Quality of code generated
 - Optimized for instruction execution time, not MIPS rating
 - Compilers are always benchmarked with the machine
- ◆ How could you “cheat” to get a high MIPS rating?

CS 141 Chien

Jan 13, 2000

Benchmarking Example II

- ◆ Power Macintosh, 450Mhz, PowerPC G4
 - Compile same program, “optimized”
 - Execute, assuming no obvious cheating
 - Experiment produces 900 MIPS rating
 - Is this faster than the Pentium II?
- ◆ => There’s no easy way to tell from this information!
- ◆ Why?
 - The unit of work has changed.
 - Pentium Instruction != PowerPC instruction
- ◆ Hard to compare MIPS across architectures, of little use for comparing architectures. Resort to execution time.

CS 141 Chien

Jan 13, 2000

Other Measures of Work

- ◆ Floating Point Operations
- ◆ Window Updates
- ◆ Frames/Polygons (rendering)
- ◆ Megabytes (communication)
- ◆ Limitations of each of these?
- ◆ How can you cheat/reduce each of these?

CS 141 Chien

Jan 13, 2000

Computer Architecture and Performance Summary

- ◆ Architecture involves a tension of programmability, compilers, implementability, and technology.
 - Optimize the design given these ever changing constraints
- ◆ Many possible measures of work / performance metrics
- ◆ Choosing is rife with potential errors
- ◆ Because it includes everything, Execution time is the safest choice.
- ◆ Still need to analyze the other influences carefully before you can draw any conclusions about the causes.

CS 141 Chien

Jan 13, 2000